



Deflate

In computing, **Deflate** (stylized as **DEFLATE**, and also called **Flate**^{[1][2]}) is a lossless data compression file format that uses a combination of LZ77 and Huffman coding. It was designed by Phil Katz, for version 2 of his PKZIP archiving tool. Deflate was later specified in RFC 1951 (1996).^[3]

Katz also designed the original algorithm used to construct Deflate streams. This algorithm was patented as U.S. patent 5,051,745 (<https://patents.google.com/patent/US5051745>), and assigned to PKWARE, Inc.^{[4][5]} As stated in the RFC document, an algorithm producing Deflate files was widely thought to be implementable in a manner not covered by patents.^[3] This led to its widespread use – for example, in gzip compressed files and PNG image files, in addition to the ZIP file format for which Katz originally designed it. The patent has since expired.

Stream format

A Deflate stream consists of a series of blocks. Each block is preceded by a 3-bit header:

- First bit: Last-block-in-stream marker:
 - 1: This is the last block in the stream.
 - 0: There are more blocks to process after this one.
- Second and third bits: Encoding method used for this block type:
 - 00: A stored (a.k.a. raw or literal) section, between 0 and 65,535 bytes in length
 - 01: A *static Huffman* compressed block, using a pre-agreed Huffman tree defined in the RFC
 - 10: A *dynamic Huffman* compressed block, complete with the Huffman table supplied
 - 11: Reserved—don't use.

The *stored* block option adds minimal overhead and is used for data that is incompressible.

Most compressible data will end up being encoded using method 10, the *dynamic Huffman* encoding, which produces an optimized Huffman tree customized for each block of data individually. Instructions to generate the necessary Huffman tree immediately follow the block header. The static Huffman option is used for short messages, where the fixed saving gained by omitting the tree outweighs the percentage compression loss due to using a non-optimal (thus, not technically Huffman) code.

Compression is achieved through two steps:

- The matching and replacement of duplicate strings with pointers.
- Replacing symbols with new, weighted symbols based on the frequency of use.

Duplicate string elimination

Within compressed blocks, if a duplicate series of bytes is spotted (a repeated string), then a back-reference is inserted, linking to the previous location of that identical string instead. An encoded match to an earlier string consists of an 8-bit length (3–258 bytes) and a 15-bit distance (1–32,768 bytes) to the beginning of the duplicate. Relative back-references can be made across any number of blocks, as long as the distance appears within the last 32 KiB of uncompressed data decoded (termed the *sliding window*).

If the distance is less than the length, the duplicate overlaps itself, indicating repetition. For example, a run of 10 identical bytes can be encoded as one byte, followed by a duplicate of length 9, beginning with the previous byte.

Searching the preceding text for duplicate substrings is the most computationally expensive part of the DEFLATE algorithm, and the operation which compression level settings affect.

Bit reduction

The second compression stage consists of replacing commonly used symbols with shorter representations and less commonly used symbols with longer representations. The method used is Huffman coding which creates an unprefix tree of non-overlapping intervals, where the length of each sequence is inversely proportional to the logarithm of the probability of that symbol needing to be encoded. The more likely it is that a symbol has to be encoded, the shorter its bit-sequence will be.

A tree is created, containing space for 288 symbols:

- 0–255: represent the literal bytes/symbols 0–255.
- 256: end of block – stop processing if last block, otherwise start processing next block.
- 257–285: combined with extra-bits, a match length of 3–258 bytes.
- 286, 287: not used, reserved and illegal but still part of the tree.

A match length code will always be followed by a distance code. Based on the distance code read, further "extra" bits may be read in order to produce the final distance. The distance tree contains space for 32 symbols:

- 0–3: distances 1–4
- 4–5: distances 5–8, 1 extra bit
- 6–7: distances 9–16, 2 extra bits
- 8–9: distances 17–32, 3 extra bits
- ...
- 26–27: distances 8,193–16,384, 12 extra bits
- 28–29: distances 16,385–32,768, 13 extra bits
- 30–31: not used, reserved and illegal but still part of the tree.

Note that for the match distance symbols 2–29, the number of extra bits can be calculated as $\left\lceil \frac{n}{2} \right\rceil - 1$.

The two codes (the 288-symbol length/literal tree and the 32-symbol distance tree) are themselves encoded as canonical Huffman codes by giving the bit length of the code for each symbol. The bit lengths are themselves run-length encoded to produce as compact a representation as possible. As an alternative to including the tree representation, the "static tree" option provides standard fixed Huffman trees. The

compressed size using the static trees can be computed using the same statistics (the number of times each symbol appears) as are used to generate the dynamic trees, so it is easy for a compressor to choose whichever is smaller.

Encoder/compressor

During the compression stage, it is the *encoder* that chooses the amount of time spent looking for matching strings. The [zlib/gzip reference implementation](#) allows the user to select from a sliding scale of likely resulting compression-level vs. speed of encoding. Options range from 0 (do not attempt compression, just store uncompressed) to 9 representing the maximum capability of the reference implementation in zlib/gzip.

Other Deflate encoders have been produced, all of which will also produce a compatible [bitstream](#) capable of being decompressed by any existing Deflate decoder. Differing implementations will likely produce variations on the final encoded bit-stream produced. The focus with non-zlib versions of an encoder has normally been to produce a more efficiently compressed and smaller encoded stream.

Deflate64/Enhanced Deflate

Deflate64, specified by PKWARE, is a proprietary variant of Deflate. It's fundamentally the same algorithm. What has changed is the increase in dictionary size from 32 KB to 64 KB, an extension of the distance codes to 16 bits so that they may address a range of 64 KB, and the length code, which is extended to 16 bits so that it may define lengths of three to 65,538 bytes.^[6] This leads to Deflate64 having a longer compression time, and potentially a slightly higher compression ratio, than Deflate.^[7] Several free and/or open source projects support Deflate64, such as 7-Zip,^[8] while others, such as [zlib](#), do not, as a result of the proprietary nature of the procedure^[9] and the very modest performance increase over Deflate.^[10]

Using Deflate in new software

Implementations of Deflate are freely available in many languages. Apps written in [C](#) typically use the [zlib](#) library (under the permissive [zlib License](#)). Apps in [Borland Pascal](#) (and compatible languages) can use [paszlib](#). Apps in [C++](#) can take advantage of the improved Deflate library in [7-Zip](#). Both [Java](#) and [.NET Framework](#) offer out-of-the-box support for Deflate in their libraries (respectively, `java.util.zip` and `System.IO.Compression` (<https://docs.microsoft.com/en-us/dotnet/api/system.io.compression.deflatestream>)). Apps in [Ada](#) can use [Zip-Ada](#) (<http://unzip-ada.sourceforge.net/>) (pure) or [ZLib-Ada](#) (<http://zlib-ada.sourceforge.net/>).

Encoder implementations

- [PKZIP](#): the first implementation, originally done by [Phil Katz](#) as part of PKZip
- [zlib](#): standard reference implementation adopted in many apps because of its open-source, permissive license. See [Zlib § Forks](#) for higher-performance forks.
- [Crypto++](#): contains a public-domain implementation in C++ aimed at reducing potential [security vulnerabilities](#). The author, Wei Dai states "This code is less clever, but hopefully

more understandable and maintainable [than zlib]".

- 7-Zip: written by Igor Pavlov in C++, this version is freely licensed and achieves higher compression than zlib at the expense of CPU usage. Has an option to use the DEFLATE64 storage format.
- PuTTY 'sshzlib.c': a standalone implementation under the MIT License by Simon Tatham, it has full decoding capability, but only supports static tree only creation
- libflate:^[11] part of Plan 9 from Bell Labs, implements deflate compression
- Hyperbac: uses its own proprietary compression library (in C++ and Assembly) with an option to implement the DEFLATE64 storage format
- Zopfli: C implementation under the Apache License by Google; achieves higher compression at the expense of CPU usage. ZopfliPNG is a variation of Zopfli for use with PNG files.
- igzip: an encoder written in the x86 assembly language, released by Intel under the MIT License. 3x faster than zlib -1. Useful for compressing genomic data.^[12]
- libdeflate:^[13] a library for fast, whole-buffer DEFLATE-based compression and decompression. Libdeflate is heavily optimized and especially on x86 processors.

AdvanceCOMP uses the higher compression ratio versions of Deflate in 7-Zip, libdeflate, and Zopfli to enable recompression of gzip, PNG, MNG and ZIP files with the possibility of smaller file sizes than zlib is able to achieve at maximum settings.^[14]

Hardware encoders

- AHA361-PCIX/AHA362-PCIX from Comtech AHA (<http://www.aha.com/>) Archived (<https://web.archive.org/web/20061208005415/http://www.aha.com/>) 2006-12-08 at the Wayback Machine. Comtech produced a PCI-X card (PCI-ID: 193f:0001) capable of compressing streams using Deflate at a rate of up to 3.0 Gbit/s (375 MB/s) for incoming uncompressed data. Accompanying the Linux kernel driver for the AHA361-PCIX is an "ahagzip" utility and customised "mod_deflate_aha" capable of using the hardware compression from Apache. The hardware is based on a Xilinx Virtex FPGA and four custom AHA3601 ASICs. The AHA361/AHA362 boards are limited to only handling static Huffman blocks and require software to be modified to add support — the cards were not able to support the full Deflate specification, meaning they could only reliably decode their own output (a stream that did not contain any dynamic Huffman type 2 blocks).
- StorCompress 300 (<http://www.indranetworks.com/SC300.html>)/MX3 (<http://www.indranetworks.com/SCMX3.html>) from Indra Networks (<http://www.indranetworks.com/>). This is a range of PCI (PCI-ID: 17b4:0011) or PCI-X cards featuring between one and six compression engines with claimed processing speeds of up to 3.6 Gbit/s (450 MB/s). A version of the cards are available with the separate brand *WebEnhance* specifically designed for web-serving use rather than SAN or backup use; a PCIe revision, the MX4E (<http://www.indranetworks.com/SCMX4E.html>) is also produced.
- AHA363-PCIe (https://web.archive.org/web/20080912222617/http://www.aha.com/show_prod.php?id=36)/AHA364-PCIe (https://web.archive.org/web/20090212202014/http://www.aha.com/show_prod.php?id=37)/AHA367-PCIe (https://web.archive.org/web/20090820184941/http://www.aha.com/show_prod.php?id=38). In 2008, Comtech started producing two PCIe cards (PCI-ID: 193f:0363/193f:0364) with a new hardware AHA3610 encoder chip. The new chip was designed to be capable of a sustained 2.5 Gbit/s. Using two of these chips, the AHA363-PCIe board can process Deflate at a rate of up to 5.0 Gbit/s (625 MB/s) using the two channels (two compression and two decompression). The AHA364-PCIe variant is an encode-only version of the card designed for out-going load balancers and instead has multiple register sets to allow 32 independent *virtual* compression channels

feeding two physical compression engines. Linux, Microsoft Windows, and OpenSolaris kernel device drivers are available for both of the new cards, along with a modified zlib system library so that dynamically linked applications can automatically use the hardware support without internal modification. The AHA367-PCIe board (PCI-ID: 193f:0367) is similar to the AHA363-PCIe but uses four AHA3610 chips for a sustained compression rate of 10 Gbit/s (1250 MB/s). Unlike the AHA362-PCIX, the decompression engines on the AHA363-PCIe and AHA367-PCIe boards are fully deflate compliant.

- Nitrox (https://web.archive.org/web/20101203144755/http://www.cavium.com/processor_security_nitrox-III.html) and Octeon (<https://github.com/zerix/Cavium-SDK-2.0/tree/master/examples/zip>) processors from Cavium, Inc. (<http://cavium.com>) contain high-speed hardware deflate and inflate engines compatible with both ZLIB and GZIP with some devices able to handle multiple simultaneous data streams.
- HDL-Deflate (<https://github.com/tomtor/HDL-deflate>) GPL FPGA implementation.
- ZipAccel-C (<https://www.cast-inc.com/compression/gzip-lossless-data-compression/zipaccel-c/>) from CAST Inc (<https://www.cast-inc.com/>). This is a Silicon IP core supporting Deflate, Zlib and Gzip compression. ZipAccel-C can be implemented in ASIC or FPGAs, supports both Dynamic and Static Huffman tables, and can provide throughputs in excess of 100 Gbit/s. The company offers compression/decompression accelerator board reference designs for Intel FPGA (ZipAccel-RD-INT (<https://www.cast-inc.com/compression/gzip-lossless-data-compression/gzip-rd-int/>)) and Xilinx FPGAs (ZipAccel-RD-XIL (<https://www.cast-inc.com/compression/gzip-lossless-data-compression/gzip-rd-xil/>)).
- Intel Communications Chipset 89xx Series (Cave Creek) for the Intel Xeon E5-2600 and E5-2400 Processor Series (Sandy Bridge-EP/EN) supports hardware compression and decompression using QuickAssist Technology. Depending on the chipset, compression and decompression rates of 5 Gbit/s, 10 Gbit/s, or 20 Gbit/s are available.^[15]
- IBM z15 CPUs incorporate an improved version of the Nest Accelerator Unit (NXU) hardware acceleration from the zEDC Express I/O expansion cards used in z14 systems for hardware Deflate compression and decompression as specified by RFC1951.^{[16][17]}
- Beginning with the POWER9 architecture, IBM added hardware support for compressing and decompressing Deflate (as specified by RFC 1951) to the formerly crypto-centric Nest accelerator (NX) core introduced with POWER7+. This support is available to programs running with AIX 7.2 Technology Level 4 Expansion Pack or AIX 7.2 Technology Level 5 Service Pack 2 through the zlibNX library.^{[18][19]}

Decoder/decompressor

Inflate is the decoding process that takes a Deflate bitstream for decompression and correctly produces the original full-size data or file.

Inflate-only implementations

The normal intent with an alternative Inflate implementation is highly optimized decoding speed, or extremely predictable RAM usage for micro-controller embedded systems.

- Assembly
 - 6502 inflate (<https://github.com/pfusik/zlib6502>), written by Piotr Fusik in 6502 assembly language.
 - SAMflate (<http://sourceforge.net/projects/samflate/>), written by Andrew Collier in Z80 assembly language with optional memory paging support for the SAM Coupé, and made available under the BSD/GPL/LGPL/DFSG licenses.

- [gunzip](https://web.archive.org/web/20160304053236/https://bitbucket.org/grauw/gunzip) (<https://web.archive.org/web/20160304053236/https://bitbucket.org/grauw/gunzip>), written by Laurens Holst in Z80 assembly language for the MSX, licensed under BSD.
- [inflate.asm](https://github.com/keirf/Amiga-Stuff) (<https://github.com/keirf/Amiga-Stuff>), a fast and efficient implementation in M68000 machine language, written by Keir Fraser and released into the Public Domain.
- C/C++
 - [kunzip](https://web.archive.org/web/20070927122958/http://www.mikekohn.net/file_formats/kunzip.php) (https://web.archive.org/web/20070927122958/http://www.mikekohn.net/file_formats/kunzip.php) by Michael Kohn and unrelated to "KZIP". Comes with C source-code under the GNU LGPL license. Used in the GIMP installer.
 - [puff.c](#) ([zlib](#)), a small, unencumbered, single-file reference implementation included in the [/contrib/puff](#) directory of the [zlib](#) distribution.
 - [tinf](http://www.ibsensoftware.com/download.html) (<http://www.ibsensoftware.com/download.html>) written by Jørgen Ibsen in ANSI C and comes with [zlib](#) license. Adds about 2k code.
 - [tinfl.c](https://code.google.com/p/miniz/source/browse/trunk/tinfl.c) (<https://code.google.com/p/miniz/source/browse/trunk/tinfl.c>) ([miniz](https://code.google.com/p/miniz/) (<https://code.google.com/p/miniz/>)), Public domain Inflate implementation contained entirely in a single C function.
- [PCDEZIP](#), Bob Flanders and Michael Holmes, published in PC Magazine 1994-01-11.
- [inflate.cl](http://opensource.franz.com/deflate/) (<http://opensource.franz.com/deflate/>) by John Foderaro. Self-standing Common Lisp decoder distributed with a GNU LGPL license.
- [inflate.s7i](http://seed7.sourceforge.net/libraries/inflate.htm) (<http://seed7.sourceforge.net/libraries/inflate.htm>)/[gzip.s7i](http://seed7.sourceforge.net/libraries/gzip.htm) (<http://seed7.sourceforge.net/libraries/gzip.htm>), a pure-Seed7 implementation of Deflate and gzip decompression, by Thomas Mertes. Made available under the GNU LGPL license.
- [pyflate](http://www.paul.sladen.org/projects/pyflate/) (<http://www.paul.sladen.org/projects/pyflate/>), a pure-Python stand-alone Deflate (gzip) and bzip2 decoder by Paul Sladen. Written for research/prototyping and made available under the BSD/GPL/LGPL/DFSG licenses.
- [deflatelua](http://lua-users.org/wiki/ModuleCompressDeflateLua) (<http://lua-users.org/wiki/ModuleCompressDeflateLua>), a pure-Lua implementation of Deflate and [gzip/zlib](#) decompression, by David Manura.
- [inflate](https://github.com/chrisdickinson/inflate) (<https://github.com/chrisdickinson/inflate>) a pure-Javascript implementation of Inflate by Chris Dickinson
- [pako](https://github.com/nodeca/pako) (<https://github.com/nodeca/pako>): JavaScript speed-optimized port of [zlib](#). Contains separate build with [inflate](#) only.

Hardware decoders

- [Serial Inflate GPU](https://web.archive.org/web/20171010000403/http://www.bitsim.com/en/our-design-model/#Blocks) (<https://web.archive.org/web/20171010000403/http://www.bitsim.com/en/our-design-model/#Blocks>) from BitSim. Hardware implementation of Inflate. Part of BitSim's *BADGE* (Bitsim Accelerated Display Graphics Engine) controller offering for embedded systems.
- [HDL-Deflate](https://github.com/tomtor/HDL-deflate) (<https://github.com/tomtor/HDL-deflate>) GPL FPGA implementation.
- [ZipAccel-D](https://www.cast-inc.com/compression/gzip-lossless-data-compression/zipaccel-d/) (<https://www.cast-inc.com/compression/gzip-lossless-data-compression/zipaccel-d/>) from CAST Inc (<https://www.cast-inc.com/>). This is a Silicon IP core supporting decompression of Deflate, Zlib and Gzip files. The ZipAccel-D IP core that can be implemented in ASIC or FPGAs. The company offers compression/decompression accelerator board reference designs for Intel FPGA ([ZipAccel-RD-INT](https://www.cast-inc.com/compression/gzip-lossless-data-compression/gzip-rd-int/) (<https://www.cast-inc.com/compression/gzip-lossless-data-compression/gzip-rd-int/>)) and Xilinx FPGAs ([ZipAccel-RD-XIL](https://www.cast-inc.com/compression/gzip-lossless-data-compression/gzip-rd-xil/) (<https://www.cast-inc.com/compression/gzip-lossless-data-compression/gzip-rd-xil/>)).
- IBM z15 CPUs incorporate an improved version of the Nest Accelerator Unit (NXU) hardware acceleration from the zEDC Express I/O expansion cards used in z14 systems for

hardware Deflate compression and decompression as specified by RFC1951.^{[16][17]}

- Beginning with the POWER9 architecture, IBM added hardware support for compressing and decompressing Deflate (as specified by RFC 1951) to the formerly crypto-centric Nest accelerator (NX) core introduced with POWER7+. This support is available to programs running with AIX 7.2 Technology Level 4 Expansion Pack or AIX 7.2 Technology Level 5 Service Pack 2 through the zlibNX library.^{[18][19]}

See also

- [List of archive formats](#)
- [List of file archivers](#)
- [Comparison of file archivers](#)

References

1. The Go Authors. "flate package - compress/flate - Go Packages" (<https://pkg.go.dev/compress/flate>). *The Go Programming Language*. Google. Retrieved 5 September 2023. "Package flate implements the DEFLATE compressed data format, described in RFC issue 1951."
2. Adobe Systems Incorporated. "PDF 32000-1:2008: Document management — Portable document format — Part 1: PDF 1.7" (https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/PDF32000_2008.pdf) (PDF). *Adobe Open Source*. Adobe. p. 23. Retrieved 5 September 2023. "FlateDecode [...] Decompresses data encoded using the zlib/deflate compression method"
3. Deutsch, L. Peter (May 1996). *DEFLATE Compressed Data Format Specification version 1.3* (<https://datatracker.ietf.org/doc/html/rfc1951#section-Abstract>). IETF. p. 1. sec. Abstract. doi:10.17487/RFC1951 (<https://doi.org/10.17487%2FRFC1951>). RFC 1951 (<https://datatracker.ietf.org/doc/html/rfc1951>). Retrieved 2014-04-23.
4. US patent 5051745 (<https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US5051745>), Katz, Phillip W., "String Searcher, and Compressor Using Same", published 1991-09-24, issued 1991-09-24, assigned to PKWare Inc.
5. David, Salomon (2007). *Data Compression: The Complete Reference* (https://books.google.com/books?id=ujnQogzx_2EC&pg=PA241) (4 ed.). Springer. p. 241. ISBN 978-1-84628-602-5.
6. "Binary Essence – Deflate64" (<https://web.archive.org/web/20170621195505/http://www.binaryessence.com/dct/imp/en000225.htm>). Archived from the original on 21 June 2017. Retrieved 22 May 2011.
7. "Binary Essence – "Calgary Corpus" compression comparisons" (<https://web.archive.org/web/20171227131819/http://www.binaryessence.com/dct/apc/en000263.htm>). Archived from the original on 27 December 2017. Retrieved 22 May 2011.
8. "-m (Set compression Method) switch" (<https://web.archive.org/web/20220409225619/https://sevenzip.osdn.jp/chm/cmdline/switches/method.htm>). *sevenzip.osdn.jp*. Archived from the original (<https://sevenzip.osdn.jp/chm/cmdline/switches/method.htm>) on 2022-04-09. Retrieved 2023-01-21.
9. History of Lossless Data Compression Algorithms – Deflate64 (http://ieeeghn.org/wiki/index.php/History_of_Lossless_Data_Compression_Algorithms#DEFLATE64)
10. zlib FAQ – Does zlib support the new "Deflate64" format introduced by PKWare? (https://www.zlib.net/zlib_faq.html#faq40)

11. "Plan 9 from Bell Labs's /n/sources/plan9/sys/src/libflate" (<https://web.archive.org/web/20060315063934/http://plan9.bell-labs.com/sources/plan9/sys/src/libflate/>). *plan9.bell-labs.com*. Lucent Technologies. Archived from the original (<http://plan9.bell-labs.com/sources/plan9/sys/src/libflate/>) on 2006-03-15.
12. "High Performance DEFLATE Compression with Optimizations for Genomic Data Sets" (<http://software.intel.com/en-us/articles/igzip-a-high-performance-deflate-compressor-with-optimizations-for-genomic-data>). *Intel Software*. 1 October 2019. Retrieved 18 January 2020.
13. "libdeflate" (<https://github.com/ebiggers/libdeflate>). *Heavily optimized library for DEFLATE/zlib/gzip compression and decompression*.
14. Mazzoleni, Andrea (21 February 2023). "amadvance/advancecomp" (<https://github.com/amadvance/advancecomp/blob/fcf71a89265c78fc26243574dda3a872574a5c02/doc/advzip.txt>). *GitHub*.
15. "Intel® Xeon® Processor E5-2600 and E5-2400 Series with Intel® Communications Chipset 89xx Series" (<https://www-ssl.intel.com/content/www/us/en/intelligent-systems/crystal-forest-server/embedded-intel-xeon-e5-2600-and-e5-2400-series-with-intel-communications-chipset-89xx.html>). Retrieved 2016-05-18.
16. "Introducing the IBM z15 - The enterprise platform for mission-critical hybrid multicloud" (<http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&supplier=877&letternum=ENUSZG19-0041>). *IBM*. 12 September 2019. Retrieved 2021-11-01.
17. Lascu, Octavian (28 April 2021). *IBM z15 (8562) Technical Guide, Page 97* (<https://books.google.com/books?id=0vr3DwAAQBAJ&dq=%22Nest+accelerator%22&pg=PA97>). IBM Redbooks. ISBN 9780738458991. Retrieved 2021-11-01.
18. "Data compression by using the zlibNX library - IBM Documentation" (<https://www.ibm.com/docs/en/aix/7.2?topic=management-data-compression-by-using-zlibnx-library>). *IBM*. Retrieved 2021-11-01.
19. "Exploitation of In-Core Acceleration of POWER Processors for AIX" (<https://community.ibm.com/community/user/power/blogs/xinya-wang1/2021/02/18/exploitation-of-nest-accelerators-and-in-core-acce>). Retrieved 2021-11-01.

External links

- PKWARE, Inc.'s `appnote.txt`, *.ZIP File Format Specification* (<http://www.pkware.com/documents/casestudies/APPNOTE.TXT>) Archived (<https://web.archive.org/web/20141205201932/http://www.pkware.com/documents/casestudies/APPNOTE.TXT>) 2014-12-05 at the Wayback Machine; Section 10, *X. Deflating – Method 8*.
 - RFC 1951 (<https://datatracker.ietf.org/doc/html/rfc1951>) – *Deflate Compressed Data Format Specification version 1.3*
 - zlib Home Page (<https://www.zlib.net>)
 - *An Explanation of the Deflate Algorithm* (<https://zlib.net/feldspar.html>) – by Antaeus Feldspar
 - *Extended Application of Suffix Trees to Data Compression* (<http://www.larsson.dogma.net/dccpaper.pdf>) Archived (<https://web.archive.org/web/20160923065920/http://www.larsson.dogma.net/dccpaper.pdf>) 2016-09-23 at the Wayback Machine – an excellent algorithm to implement Deflate by Jesper Larsson
 - Zip Files: History, Explanation and Implementation (<https://www.hanshq.net/zip.html>) – walk-through of a Deflate implementation
-

Retrieved from "<https://en.wikipedia.org/w/index.php?title=Deflate&oldid=1265222833>"